

## EE 422C Project 2

### Object Oriented Mastermind

*See Canvas for due dates.*

This is a Pair Programming Assignment. You must work with one other person on this assignment using the pair programming technique. Review the paper on pair programming posted on the class web page. Your partner and you will both get the same grade for the assignment. Make sure both names are submitted with the design and code. If you begin working with one partner and do not wish to finish the assignment with that partner then you must complete the assignment individually. Your TA must be so informed before the due date.

#### General Assignment Requirements

The purpose of this assignment is to design and implement an OO program with multiple classes to play a text based version of the board game called Mastermind. You are free to use whatever classes and methods from the JAVA library you wish to use.

First thing to do is - Read the Wikipedia article on the game of Mastermind at:

[http://en.wikipedia.org/wiki/Mastermind\\_\(board\\_game\)](http://en.wikipedia.org/wiki/Mastermind_(board_game))

Here is a good example of what an interactive GUI for Mastermind would look like:

<http://www.web-games-online.com/mastermind/index.php>

The version of the game you implement will have the following properties.

- The computer will randomly generate the secret code.
- The player will try to guess the secret code.
- The player has 12 guesses to guess the code correctly. This number should be easily changeable by modifying your code.
- If the player does not guess the code correctly in 12 or fewer guesses they lose the game.
- The secret code consists of 4 colored pegs in specific position order.
- The valid colors for the pegs are blue, green, orange, purple, red, and yellow. Capital letters will be used in the examples to indicate colors. B for blue, R for red, and so forth. The number of colors should be changeable for full credit. The selection of the number of colors, like the number of guesses and the number of pegs, should be changeable through the provided class GameConfiguration.java.
- The results of a guess are displayed with black and white pegs. The Wikipedia article refers to the results as feedback.
- A black peg indicates one of the pegs in the player's guess is the correct color and in the correct position.
- A white peg indicates one of the pegs in the player's guess is the correct color, but is out of position.
- A peg in the guess will generate feedback of either: 1 black peg, 1 white peg, or no pegs. A single peg in the guess cannot generate more than 1 feedback peg.
- The order of the feedback does not give any information about which pegs in the guess generated which feedback pegs.
- The player's guesses must be error checked to ensure they are the correct length and only contain valid characters.

The output of the game should be a simple text based display on the console, similar to the following example:

### Sample user dialogue

Console output is in typewriter font, and user input is **bold-underlined**. Do not change the text or format. If you pipe your output to a file instead of the console, the user input part will be missing in the file. We will therefore allow extra of missing newlines in the places where the user inputs text.

Initial greeting.

Welcome to Mastermind. Here are the rules.

This is a text version of the classic board game Mastermind.  
The computer will think of a secret code. The code consists of 4 colored pegs.

The pegs MUST be one of six colors: blue, green, orange, purple, red, or yellow. A color may appear more than once in the code. You try to guess what colored pegs are in the code and what order they are in. After you make a valid guess the result (feedback) will be displayed.

The result consists of a black peg for each peg you have guessed exactly correct (color and position) in your guess. For each peg in the guess that is the correct color, but is out of position, you get a white peg. For each peg that is fully incorrect, you get no feedback.

Only the first letter of the color is displayed. B for Blue, R for Red, and so forth.

When entering guesses you only need to enter the first character of each color as a capital letter.

You have 12 guesses to figure out the secret code or you lose the game. Are you ready to play? (Y/N): **Y**

Generating secret code .... (for this example the secret code is YRBY)

You have 12 guesses left.

What is your next guess?

Type in the characters for your guess and press enter.

Enter guess: **0000**

0000 -> Result: No pegs

You have 11 guesses left.

What is your next guess?

Type in the characters for your guess and press enter.

Enter guess: **oooo**  
oooo -> INVALID GUESS

What is your next guess?  
Type in the characters for your guess and press enter.  
Enter guess: **kkkk**  
kkkk -> INVALID GUESS

What is your next guess?  
Type in the characters for your guess and press enter.  
Enter guess: **RRRRR**  
RRRRR -> INVALID GUESS

What is your next guess?  
Type in the characters for your guess and press enter.  
Enter guess: *(no text, or spaces)*  
-> INVALID GUESS

What is your next guess?  
Type in the characters for your guess and press enter.  
Enter guess: **RRRR**

RRRR -> Result: 1 black peg

You have 10 guesses left.  
What is your next guess?

*(Etc. etc.,...)*

What is your next guess?  
Type in the characters for your guess and press enter.  
Enter guess: **YRBY**  
YRBY -> Result: 4 black pegs - You win!!  
(Alternately: (Sorry, you are out of guesses. You lose, boo-hoo.)

Are you ready for another game (Y/N) : **N**

Part of the assignment grade will be determined by how easily your program could be altered to allow a different maximum number of guesses, or a different number of pegs in the code and how easily new colors could be added, assuming they start with a different letter than other existing colors. (Up to 10) For example how easily would it be to change the code to have 5 pegs and allow pegs to be the color Maroon?

One of the criteria of the assignment is to break the problem up into smaller classes even if you think the problem could be solved more easily with ONE BIG CLASS. For this assignment you should have more than 1 class.

The top-level class of your program must be called **Game**. It must have a constructor that takes a boolean value as its parameter. The boolean value is used for running the game in the testing mode until you are finished. If it is true, then the secret code is revealed throughout the game. The Game class must also have a method named `runGame` that carries out the actual games. **Your main method creates and uses a new Game object for each game played.**

When finished turn in a file named **Project1\_EID1\_EID2.zip** that contains all the files needed for your program to be compiled and run. Include all the source code for all the classes you created.

Recall that when designing a program in an object oriented way, you should consider implementing a class for each type of entity you see in the problem. For example in Mastermind there is a game board, pegs, colors, codes (the secret code and the player's guesses can both be considered the same data type, a code), feedback results, a computer player, a human player, and an over-all game runner. Some things are so simple you may choose not to implement a class for them. For example the computer player doesn't do anything more than pick the secret code and answer the guesses. Maybe that is simple enough for the Game or Board class to do. Also you may use some pre-existing type, primitive or a class, to represent things. E.g. the guesses and results could be String objects.

After deciding what classes you need, implement them one at a time, simplest ones first. Test a class thoroughly before going on to the next class. You may need to alter your design as you implement and test classes. Remember Design a little - code a little - test a little. Repeat.

I recommend you work on this incrementally. Start with a design and try to get that to work. Have a working program at all times and add to it as you implement more features. This will avoid the assignment becoming an all or nothing affair. Even if you don't finish you will have a working version with some functionality ready to turn in.

## Design deliverables

The design to be turned in on paper will consist of the following:

- A **UML use case model diagram**
- A **UML class diagram**
- 2 **UML sequence diagrams (or flowcharts)** that express the high level algorithm descriptions for a) the game runner, and b) formulating the reply to a given guess

## Tips and Hints

The hardest thing algorithmically about this assignment is how to generate a result given a secret code and a guess. It is important to realize black pegs should be generated first and that a given peg from the secret code and the guess cannot be reused once matched.

Here are some examples:

Code: RRRR  
Guess: BBBB

This one is easy. There aren't any B's in the code, so the guess of BBBB would generate a result with no pegs.

Code: RRRR  
Guess: RBBY

The R in the first position of the guess matches up exactly with the R in the first position of the code. This generates one black peg. The key is once a peg has generated a peg it cannot be used to generate any more. So the R in the first position of the code and the first position of the guess cannot generate any more pegs. The guess of RBBY generates a result with 1 black peg and 0 white pegs.

Here is another example.

Code: RBYR  
Guess: BBRG

This code and guess would give a result of 1 black peg and 1 white peg. Black pegs take precedence so the B in the second position of the code and guess result in a black peg. One useful way of solving this problem is to determine the black pegs and scratch out or replace the pegs that were used. So given the Code of RBYR and the Guess of BBRG there could be a temporary result of

Code: R-YR  
Guess: B-RG

To get the number of white pegs in the result the remaining characters can be compared for matches, independent of position. If a match is found then those pegs should be scratched out as well. The B in the first position of the guess doesn't match any characters in the code. The "-" in the second position of the guess should be skipped. The r in the third position of the guess matches the R in the first position of the code. This is a peg in the guess that is the right color, but in the wrong position so it generates 1 white peg. Once used the R in the first position of the code and the R in the third position of the guess can be reused so they should be scratched out.

Code: --YR  
Guess: B--G

The G in the fourth position of the guess doesn't match anything so the result would be 1 black peg and 1 white peg. To summarize

Code: RBYR  
Guess: BBRG

generates a result of 1 black peg and 1 white peg.

In correctly guessing the code, it is helpful for you to be able to review the history of the guesses and replies. This represents the state of the Board. Therefore, you should implement a way to

have the program output the history in order from the beginning whenever you want to see it. So, when the user types in “history” for the guess, the program would then display that history.

What is your next guess?

Type in the characters for your guess and press enter.

Enter guess: history

... (the history of all guesses and replies for this game are displayed in tabular format – first the guess, then two tab characters, and then the computer’s reply. )

**Warning** - You may not acquire, from any source (e.g., another student or an internet site), a partial or complete solution to this problem. Except for your partner, you may not show another student your solution to this assignment. You may not have another person (TA, current student, former student, tutor, friend, anyone) “walk you through” how to solve this assignment. Review the class policy on cheating from the syllabus.

**Tip** – we may come back to this assignment later in the semester to put a fancy GUI on it, so design it with that in mind. I.e. encapsulate the user interface so it can be replaced.

**Tip** - Each team is required to use the git repository for interim and final versions of your code. The rule of thumb is that you should commit at least once per working day when code is being generated, preferably at the end of each working session where code is being changed. Please see the git tutorial provided by your TA for more information on how to use that.

#### **CHECKLIST – Did you remember to:**

- ☐ Follow pair programming guidelines, including recording each partner’s share of the work?
- ☐ Re-read the requirements after you finished your program to ensure that you meet all of them?
- ☐ Make up your own testcases?
- ☐ Use the Git repository regularly?
- ☐ Make sure that all your submitted files have the appropriate header file?
- ☐ Upload your solution to Canvas, remembering to include ALL your files in one zip file name Project1\_EID1\_EID2.zip?
- ☐ Download your uploaded solution into a fresh directory and re-run all testcases?

*Adapted from an assignment written by Mike Scott. Thanks also to Herb Kreisner.*